Università degli Studi di Bologna
Facoltà di Ingegneria

Progettazione di Applicazioni Web T

Esercitazione 7
Hibernate: un primo esercizio di base

Agenda

- **Libreria Hibernate**
  - **progetto d'esempio** relativo alla «gestione dei dati degli studenti»
  - veloce ripasso di alcuni concetti chiave già visti a lezione
  - **proposta esercizi da svolgere in autonomia** relativo alla «gestione dei corsi universitari», a partire dal progetto d'esempio

Progetto di esempio

- Il file ***07_PAWeb.zip*** contiene lo scheletro di un semplice progetto di esempio basato sull'uso di Hibernate
- Importare il progetto come visto nelle precedenti esercitazioni, senza estrarre l'archivio su file system (lo farà Eclipse)
  - *File → Import → General → Existing Projects into Workspace → Next → Select archive file*
- Progetto (nella sua versione «completa») per la gestione del database di corsi universitari
  - elenco dei **corsi attivi**
  - elenco degli **studenti**
  - **mapping tra corsi e studenti che li frequentano**
- **!** L'esercizio guidato si concentra su «studenti» e riprende la classe **StudentRepository** già vista per le metodologie «forza bruta» e pattern DAO

Hibernate

- Uso di librerie (ad esempio, Hibernate) rende più «facile» l'accesso a DBMS
  - uso di API che rendono più semplice e meno *error-prone* l'accesso a database
  - accesso a database trasparente rispetto al particolare DBMS in uso
  - riconfigurazione facile e veloce se si vuole utilizzare un DBMS diverso da quello precedentemente in uso

Hibernate: hibernate.cfg.xml

- Nel package **it.unibo.paw.hibernate**, accesso a DBMS con hibernate
- La classe **Student** è un Java Bean, identico a **StudentDTO**
- Il file **hibernate.cfg.xml** specifica tutto il necessario per identificare ed accedere ad un database
  - driver JDBC, URI, username, password, dialetto SQL...
  - posizionato di default nella directory **src**

```
<property name="connection.driver_class">
   com.ibm.db2.jcc.DB2Driver
</property>
<property
   name="connection.url">jdbc:db2://diva.deis.unibo.it:50000/tw_stud
</property>
<property name="connection.username">username</property>
<property name="connection.password">password</property>
```

Hibernate: Student.hbm.xml

- Inoltre il file **hibernate.cfg.xml** identifica altri file XML di configurazione che, a loro volta, dichiarano il mapping tra classi Java Bean e specifiche tabelle presenti nel database

```
<mapping resource="it/unibo/paw/hibernate/Student.hbm.xml"/>
```

- Il file **Student.hbm.xml** specifica il mapping tra oggetto Java Bean e tabella del database
  - posizionato di default nella directory corrispondente al package del Java Bean
  - nome della tabella
  - primary key della tabella
  - mapping tra proprietà del Java Bean e colonna della tabella

```
<class name="it.unibo.paw.hibernate.Student"
   table="students">
    <id name="id" column="id" />
    <property name="firstName" column="firstName"/>
    <property name="lastName" column="lastName"/>
    <property name="birthDate" column="birthDate"/>
</class>
```

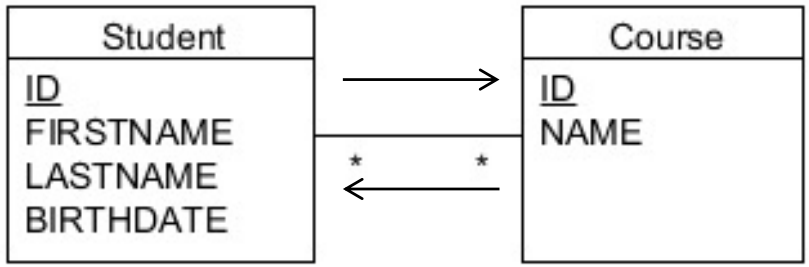
Hibernate: classe HibernateTest

- La classe **HibernateTest** crea e rende persistente due Java Bean Student e poi effettua query sul database
  1) si ottiene un sessione
  2) si inizia una transazione
  3) si richiede la persistenza di uno o più Java Bean
  4) si termina la transazione tramite commit
  5) eventualmente si inizia una nuova transazione (punto 2)
  6) infine si termina la sessione (vedi **finally**)
  - in caso di errore si richiede il rollback della transazione (vedi **catch**)

- Dopo il commit, viene richiesto **l'elenco degli studenti che hanno cognome "Gialli"**
  - Hibernate supporta query di tipo HQL ed SQL
    - con **HQL** il nome della tabella viene identificato tramite mapping via Java Bean e XML
    - con **SQL** è necessario specificare il nome della tabella (come in JDBC)
  - in alternativa è possibile effettuare la query specificando criteri di restrizione, in modo object-oriented al 100%
    - Criteria su classe Student (e quindi corrispondente tabella students), specificando la Restriction che il lastName deve essere Gialli

Ora a voi (1)

- Partendo dal progetto di esempio «studenti», e considerando il diagramma UML di seguito riportato relativo alla «gestione dei corsi universitari»



  estendere le funzionalità dell'applicazione esistente in modo tale da fornire una soluzione alla gestione della persistenza basata su Hibernate in grado di “mappare” efficientemente il modello di dominio rappresentato dai Java Bean **Student, Course** con le corrispondenti tabelle derivate dalla progettazione logica applicata al diagramma UML, ovvero **Student**, **Course** e **S-CMapping**, contenute nel DB **TW_STUD**

- **N.B. La soluzione deve: 1) procedere con la creazione esplicita delle tabelle da applicazione, con esplicitazione degli opportuni vincoli di PK e FK; 2) mantenere il mapping N-M tra i due Java Bean e rispettare i sensi di percorrenza indicati dalle frecce nell’UML**

Ora a voi (2)

- Creare **il nuovo file xml di mapping necessario** per i corsi
- Modificare il **file di configurazione hibernate.cfg.xml** per includere anche i due nuovi file di configurazione appena creati

- **Estendere la classe HibernateTest** per ottenere
  - l'**elenco dei corsi** frequentati da uno studente (identificato tramite l'ID dello studente)
  - **l'elenco degli studenti** che frequentano un corso (identificato tramite l'ID del corso)

Ora a voi (3)

- **Come si mappa una proprietà che rappresenta istanze di una altra classe di dominio Java?**
  - Ad esempio, la collezione di oggetti Course nel JavaBean Student e la collezione di oggetti Student nel JavaBean Course?
- Nel file di mapping si definisce un **elemento apposito *Set***

```
<hibernate-mapping>
    <class name="it.unibo.paw.hibernate.Student" table="students">
        <id name="id" column="id">
            <!-- <generator class="native" /> -->
        </id>
        <property name="firstName" column="firstName" />
        <property name="lastName" column="lastName" />
        <property name="birthDate" column="birthDate" />

        <set name="courses" table="courses_students">
            <key column="idStudent" />
            <many-to-many column="idCourse"
                class="it.unibo.paw.hibernate.Course" />
        </set>
    </class>
</hibernate-mapping>
```

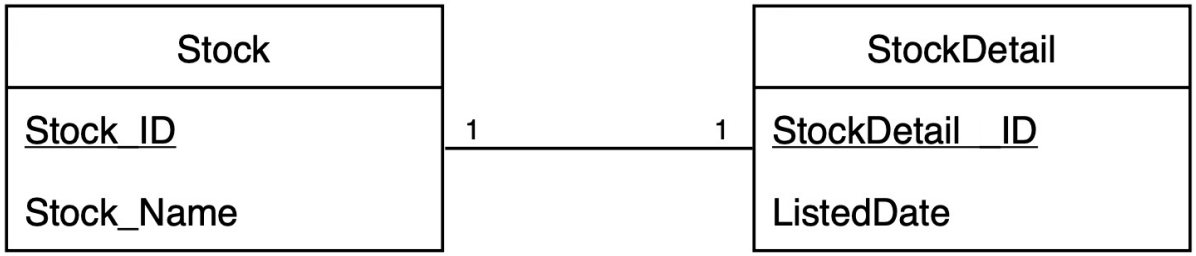
```
<hibernate-mapping>
    <class name="it.unibo.paw.hibernate.Course" table="courses">
        <id name="id" column="id">
            <!-- <generator class="native"/> -->
        </id>
        <property name="name" column="name" />

        <set name="students" table="courses_students" inverse="true">
            <key column="idCourse" />
            <many-to-many column="idStudent"
                class="it.unibo.paw.hibernate.Student" />
        </set>
    </class>
</hibernate-mapping>
```

- **Generalizziamo il concetto sopra illustrato riportando di seguito esempi dei diversi tipi di mapping: 1-1, 1-N, N-M ☺**
- N.B. Si rimanda alla documentazione ufficiale Hibernate per completare la panoramica!

Mapping relazioni 1-1: esempio

N.B. Con un piccolo abuso di notazione, nei diagrammi UML la sottolineatura di un attributo non indica la staticità di tale attributo bensì il suo vincolo di univocita alla E/R.

Consideriamo il seguente diagramma UML riguardanti gli stock in un magazzino e ai relativi dettagli



Le classi Stock e StockDetail Saranno le seguenti:

```java
public class Stock
    implements java.io.Serializable {

    private Integer stock_ID;
    private String stock_Name;
    private StockDetail stockDetail;

    //constructor & getter and setter methods
}
```

```java
public class StockDetail
    implements java.io.Serializable {

    private Integer stockDetail_ID;
    private Date listedDate;
    private Stock stock;

    //constructor & getter and setter methods
}
```

Mapping relazioni 1-1: esempio

Stock.hbm.xml:

```
<hibernate-mapping>
    <class name="it.unibo.paw.hibernate.Stock" table="stock">
        <id name="stock_ID" type="java.lang.Integer">
            <column name="STOCK_ID" />
        </id>
        <property name="stock_Name" type="string">
            <column name="STOCK_NAME" length="20" />
        </property>
        <one-to-one name="stockDetail"
            class="it.unibo.paw.hibernate.StockDetail">
        </one-to-one>
    </class>
</hibernate-mapping>
```

Dove:

- name indica il nome della proprietà all'interno della classe (Stock);
- class indica l'altra classe nella relazione (StockDetail).

Mapping relazioni 1-1: esempio

StockDetail.hbm.xml:

```
<hibernate-mapping>
    <class name="it.unibo.paw.hibernate.StockDetail" table="stock_detail">
        <id name="stockDetail_ID" type="java.lang.Integer">
            <column name="STOCKDETAIL_ID" />
        </id>
        <property name="listedDate" type="date">
            <column name="LISTED_DATE" length="10"/>
        </property>
        <one-to-one name="stock"
            class="it.unibo.paw.hibernate.Stock">
        </one-to-one>
    </class>
</hibernate-mapping>
```

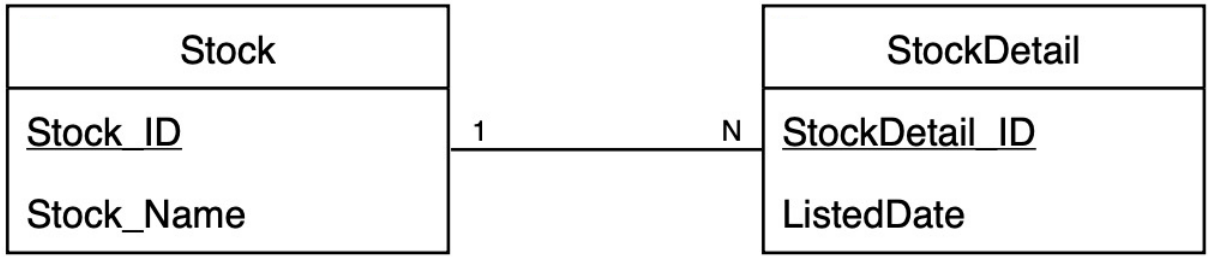
Dove:

- name indica il nome della proprietà all'interno della classe (StockDetail);
- class indica l'altra classe nella relazione (Stock).

Mapping relazioni 1-N: esempio

N.B. Con un piccolo abuso di notazione, nei diagrammi UML la sottolineatura di un attributo non indica la staticità di tale attributo bensì il suo vincolo di univocita alla E/R.

Consideriamo il seguente diagramma UML riguardanti gli stock in un magazzino e ai relativi dettagli



Le classi Stock e StockDetail Saranno le seguenti:

```java
public class Stock
    implements java.io.Serializable {

    private Integer stock_ID;
    private String stock_Name;
    private Set<StockDetail> stockDetails;

    //constructor & getter and setter methods
}
```

```java
public class StockDetail
    implements java.io.Serializable {

    private Integer stockDetail_ID;
    private Date listedDate;
    private Stock stock;

    //constructor & getter and setter methods
}
```

Mapping relazioni 1-N: esempio

Stock.hbm.xml:

```
<set name="stockDetails" table="stock_detail"
        inverse="true" lazy="true" fetch="select">
    <key>
        <column name="STOCK_ID" not-null="true" />
    </key>
    <one-to-many class="it.unibo.paw.hibernate.StockDetail"/>
</set>
```

- **set**: indica la presenza di una collezione di oggetti
  - Attributo **name**: indica il nome del set all'interno della classe;
  - Attributo **table**: indica la tabella associata nella relazione;
  - **key**: indica la chiave esterna
- Attributo **Name** di Column: nome della chiave della tabella stessa;
- **one-to-many**: relazione
- Attributo **class**: indica l'altra classe associata nella relazione

Mapping relazioni 1-N: esempio

StockDetail.hbm.xml:

```
<many-to-one name="stock" class="it.unibo.paw.hibernate.Stock" fetch="select">
    <column name="STOCK_ID" not-null="true" />
</many-to-one>
```

- **many-to-one**: tipo di relazione
  - Attributo **name**: indica il nome della proprietà all'interno della classe;
  - Attributo **class**: indica l'altra classe associata nella relazione.;
  - **column**: colonna di riferimento della tabella esterna
  - Attributo **name**: nome della primary key detta tabella esterna.

Mapping relazioni N-M: esempio

N.B. Con un piccolo abuso di notazione, nei diagrammi UML la sottolineatura di un attributo non indica la staticità di tale attributo bensì il suo vincolo di univocita alla E/R.

Consideriamo il seguente diagramma UML riguardanti gli stock in un magazzino e ai relativi dettagli

Le classi Stock e Category Saranno le seguenti:

```java
public class Stock
    implements java.io.Serializable {

    private Integer stock_ID;
    private String stock_Name;
    private Set<Category> categories;

    //constructor & getter and setter methods
}
```

```java
public class Category
    implements java.io.Serializable {

    private Integer categoryId;
    private String name;
    private Set<Stock> stocks;

    //getter, setter and constructor
}
```

Mapping relazioni N-M: esempio

Stock.hbm.xml:

```
<set name="categories" table="stock_category"
    inverse="false" lazy="true" fetch="select" cascade="all" >
    <key column="STOCK_ID"/>
    <many-to-many column="CATEGORY_ID" class="it.unibo.paw.hibernate.Category"/>
</set>
```

- I significati di attributi ed elementi sono gli stessi visti in precedenza
- L'attributo **table** di set tuttavia si riferisce alla **"tabella di mapping"** (ovvero la tabella DB derivata dal passo di progettazione logica applicato alla associazione N-M)

Category.hbm.xml:

```
<set name="stocks" table="stock_category"
     inverse="true" lazy="true" fetch="select">
    <key column="CATEGORY_ID"/>
    <many-to-many column="STOCK_ID" class="it.unibo.paw.hibernate.Stock"/>
</set>
```

Ulteriori link

- **API ufficiali di Hibernate (versione 4.3) al seguente link**
  - http://docs.jboss.org/hibernate/core/4.3/javadocs/
- **Altri link utili**
  - https://www.mkyong.com/tutorials/hibernate-tutorials/
  - https://www.mkyong.com/hibernate/hibernate-one-to-many-relationship-example/
  - https://www.mkyong.com/hibernate/hibernate-many-to-many-relationship-example/